```
ABSTRACT CLASS Policy
        PROTECTED policyType
        PROTECTED basePremium
        PROTECTED coverageAmount
        PROTECTED termYears

        CONSTRUCTOR(policyType, basePremium, coverageAmount, termYears)
                SET this.policyType = policyType
                SET this.basePremium = basePremium
                SET this.coverageAmount = coverageAmount
                SET this.termYears = termYears

        ABSTRACT METHOD calculatePremium(assumptions: AssumptionSet): NUMBER
END CLASS


CLASS LifePolicy EXTENDS Policy
        // Life-specific inputs

        CONSTRUCTOR(basePremium, coverageAmount, termYears)
                CALL super("Life", basePremium, coverageAmount, termYears)

        OVERRIDE METHOD calculatePremium(assumptions)
                expectedClaim = coverageAmount * assumptions.mortalityRate
                grossPremium = basePremium * (1 + assumptions.expenseLoad)
                RETURN grossPremium + (basePremium * assumptions.mortalityRate)
END CLASS


CLASS AssumptionSet
        PRIVATE mortalityRate
        PRIVATE expenseLoad
        PRIVATE interestRate

        CONSTRUCTOR(mortalityRate, expenseLoad, interestRate)
                VALIDATE ranges (all values must be >= 0)
                SET this.mortalityRate = mortalityRate
                SET this.expenseLoad = expenseLoad
                SET this.interestRate = interestRate
END CLASS


// ------------------- Math Engine -------------------
CLASS ActuarialCalculator

        METHOD expectedClaim(policy: Policy, assumptions: AssumptionSet): NUMBER
                RETURN policy.coverageAmount * assumptions.mortalityRate

        METHOD presentValueOfBenefits(policy, assumptions): NUMBER
                i = assumptions.interestRate
                claim = expectedClaim(policy, assumptions)
                annuityFactor = (1 - (1 + i)^(-policy.termYears)) / i
                RETURN claim * annuityFactor

        METHOD netPremiumPerYear(policy, assumptions): NUMBER
                pv = presentValueOfBenefits(policy, assumptions)
```

```
        RETURN pv / policy.termYears
END CLASS


// ------------------- Authentication -------------------
CLASS SSOService
    PRIVATE validTokens = {"ADMIN123"}

    METHOD authenticate(token: STRING): BOOLEAN
        RETURN token IN validTokens
END CLASS


// ------------------- Main Application -------------------
CLASS ActuarialApp

    METHOD run()
        DISPLAY "Enter SSO token:"
        token = INPUT

        IF NOT SSOService.authenticate(token) THEN
            DISPLAY "Access denied."
            EXIT
        ENDIF

        DISPLAY "Enter base premium:"
        base = INPUT_NUMBER

        DISPLAY "Enter coverage amount:"
        coverage = INPUT_NUMBER

        DISPLAY "Enter term (years):"
        term = INPUT_INT

        DISPLAY "Enter mortality rate (e.g., 0.002):"
        qx = INPUT_NUMBER

        DISPLAY "Enter expense load (e.g., 0.10):"
        load = INPUT_NUMBER

        DISPLAY "Enter interest rate (e.g., 0.04):"
        i = INPUT_NUMBER

        assumptions = NEW AssumptionSet(qx, load, i)
        policy = NEW LifePolicy(base, coverage, term)
        calc = NEW ActuarialCalculator()

        premium = policy.calculatePremium(assumptions)
        pv = calc.presentValueOfBenefits(policy, assumptions)
        net = calc.netPremiumPerYear(policy, assumptions)
        expClaim = calc.expectedClaim(policy, assumptions)

        DISPLAY "----- Results -----"
        DISPLAY "Expected Claim: ", expClaim
        DISPLAY "Gross/Loaded Premium: ", premium
        DISPLAY "PV of Benefits: ", pv
```

```
                DISPLAY "Net Premium / Year: ", net
END CLASS
```